动态规划引入

动态规划引入

动态规划主要用于**优化问题求解**，即求出问题的**最优解**

■ 与分治法异同

❖**相同点：**都是通过将求解问题划分为若干小规模的子问题，进行求解，再合并子问题的最优解，来解决整个问题的解

❖**不同点：**

1)**分治法**是将大问题划分为相似但小规模的子问题，递归地求解子问题，然后将子问题的解合并，分治法是递归地求解了**所有**划分出的子问题，包括了那些**重复**的子问题！因此仅适用于子问题**相互独立**的场景。

2)当分解问题**非独立**时，更应该采用**动态规划**：将求解过的子问题答案都保存起来，从而保证每个分解出的子问题只会被求解一次。

动态规划步骤

Step1、2、3
是基础

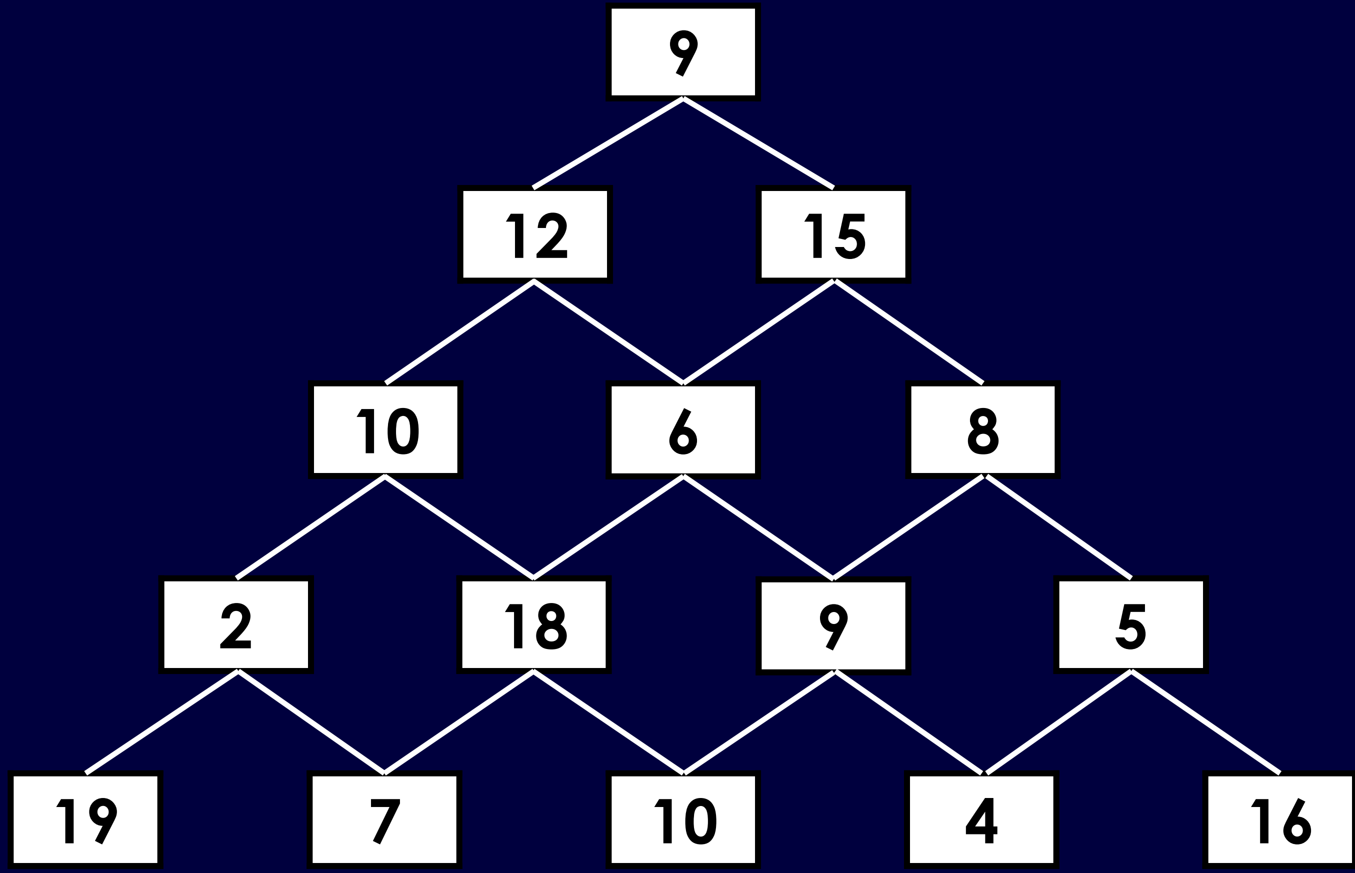
四个步骤

- Step1：描述最优解的结构特征
- Step2：递归地定义一个最优解的值
- Step3：自底向上计算一个最优解的值
- Step4：从已计算的信息中构造一个最优解

step3中记录子问题的答案，在下次遇到该子问题时可以直接查询获取答案，如要构造最优解则还需要维护额外的附加信息，如求最短路径，除了路径长度外还需储存经过的路径信息。

例：数塔问题

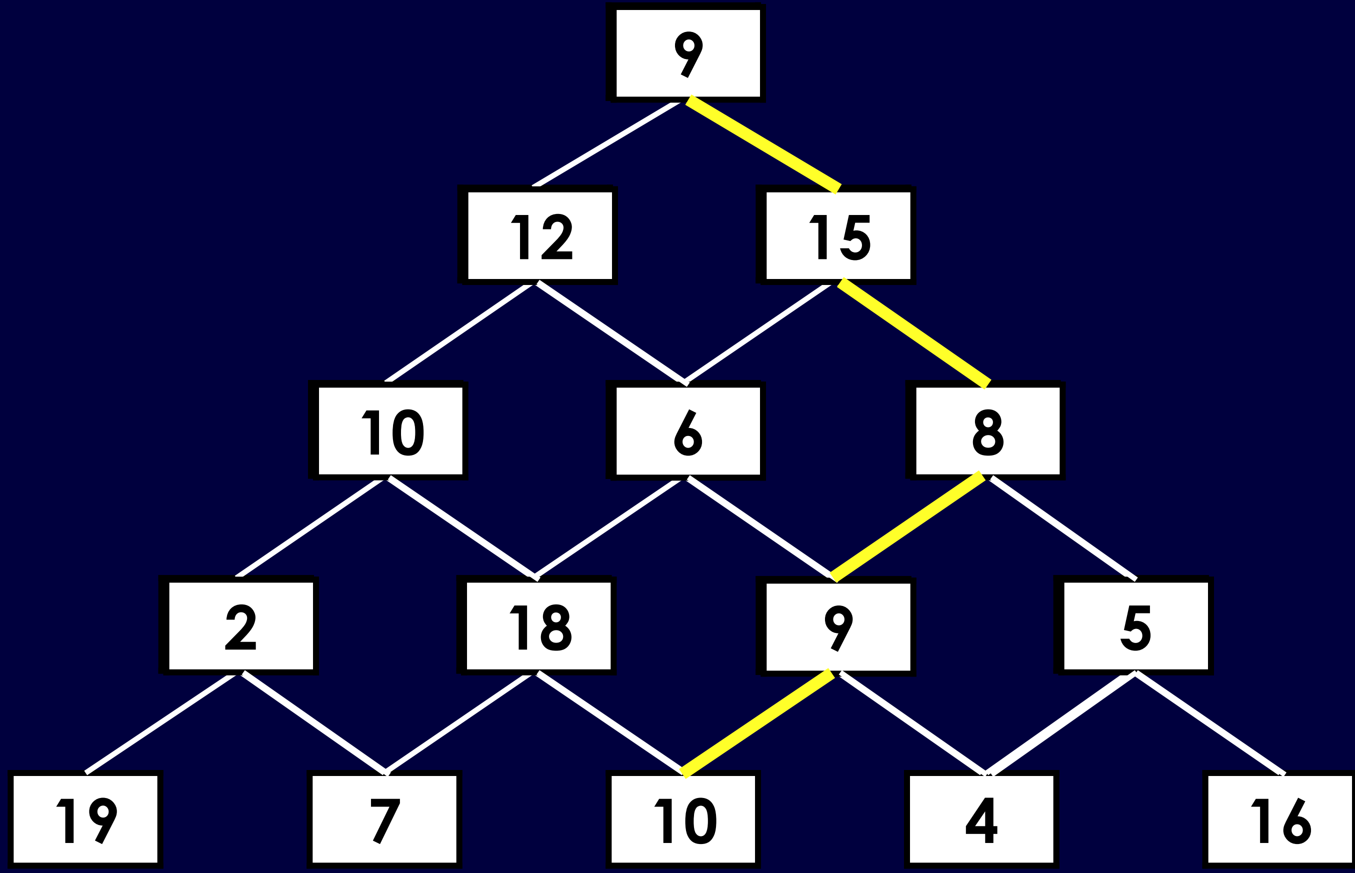
如图，有一个数塔，从顶部出发一直走到底层，或者从底层走到顶层，在每一个节点可以选择向左或者向右走。要求找出一条路径，使得所走路径上的节点数字之和最大。



贪心法求解

- 思路1：贪心思想。

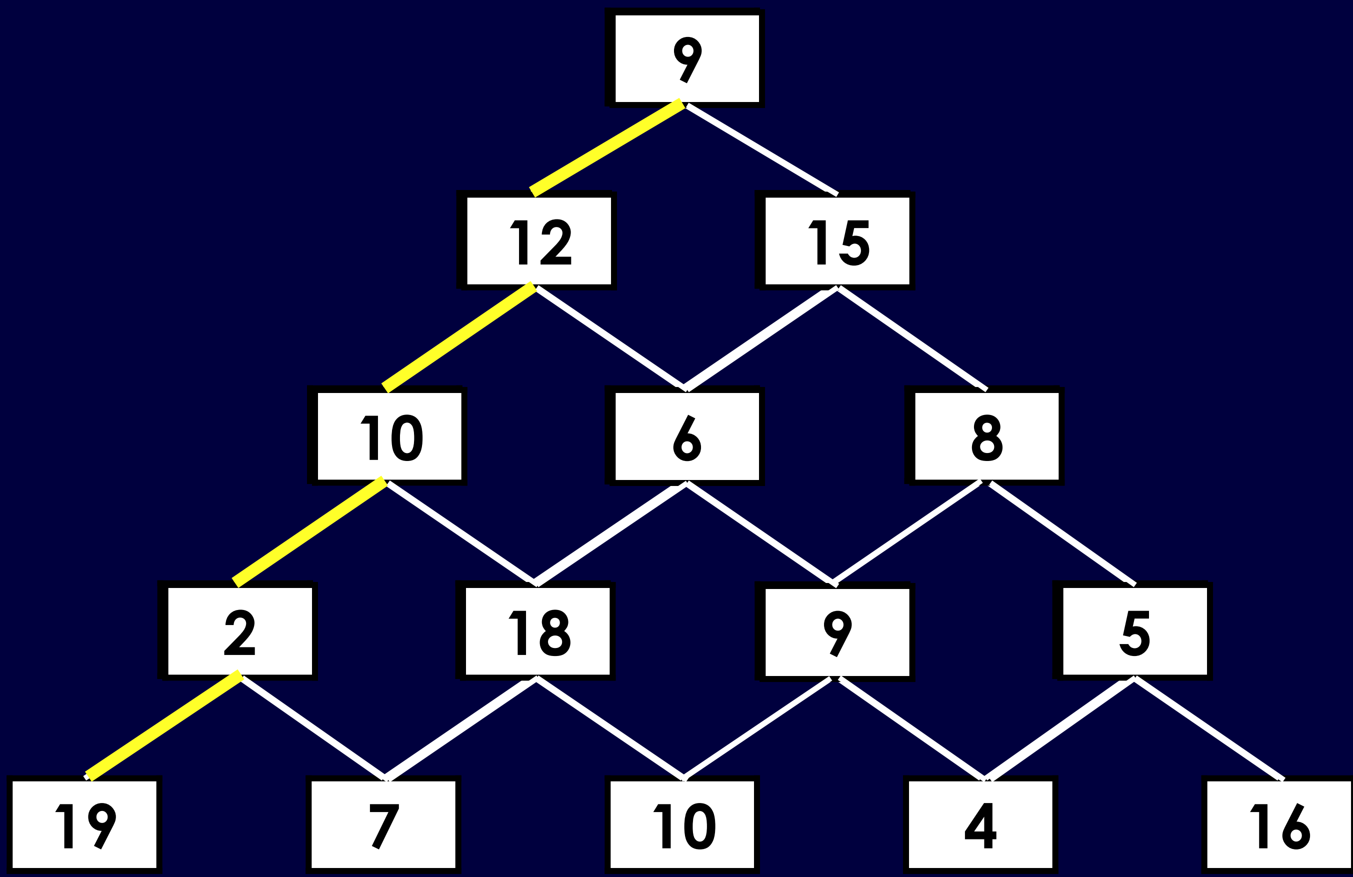
如果自顶向下：路径为9+15+8+9+10=51，可能不是真正的最大和



贪心法求解

- 思路1：贪心思想。

如果自底向上：路径为19+2+10+12+9=52，是最优解吗？



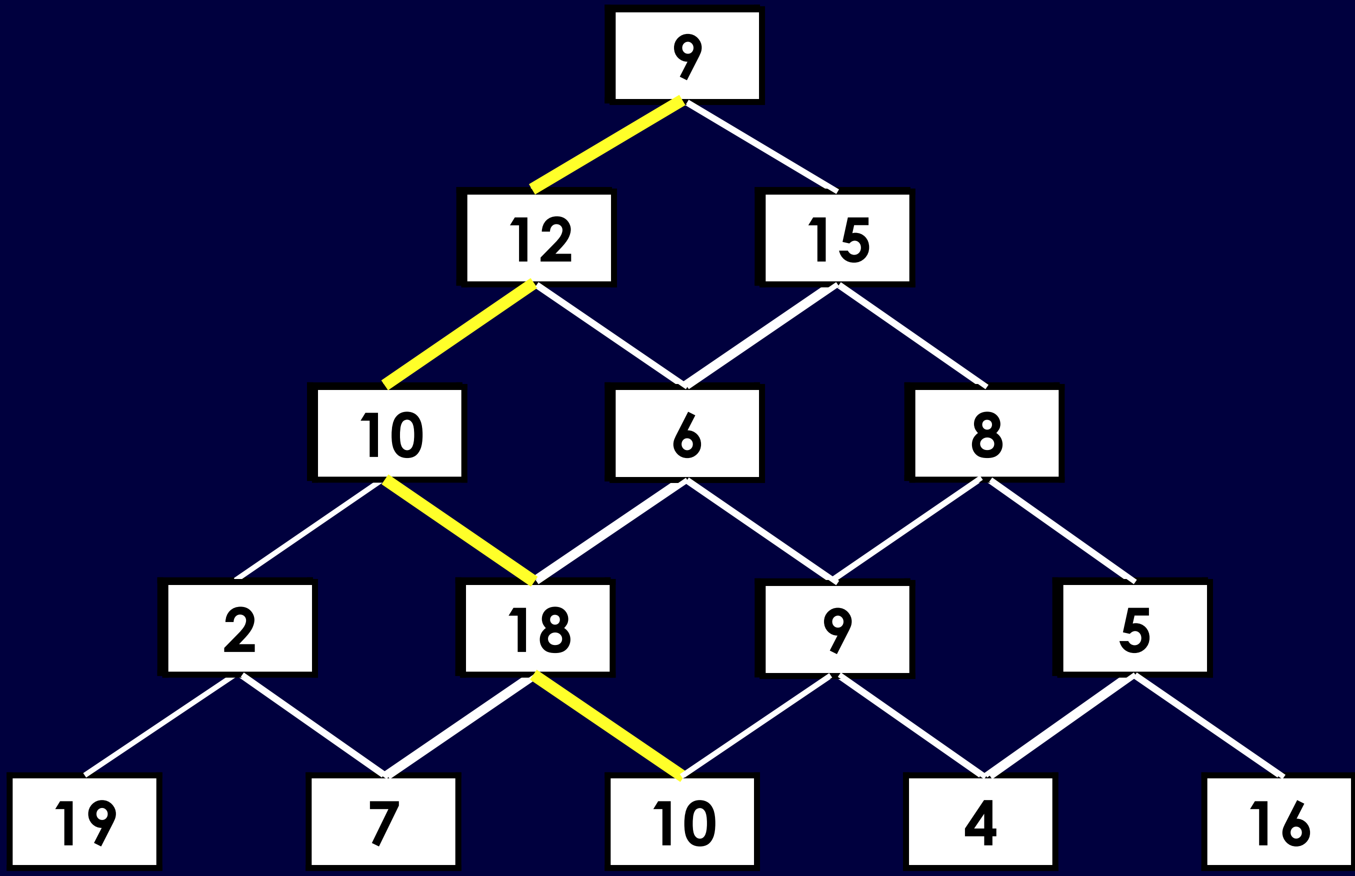
贪心法求解

❖思路1：贪心思想。

真正的最优解：路径为10+18+10+12+9=59

总结：找到最大和的前提是能看到数塔全貌

贪心算法只考虑到局部



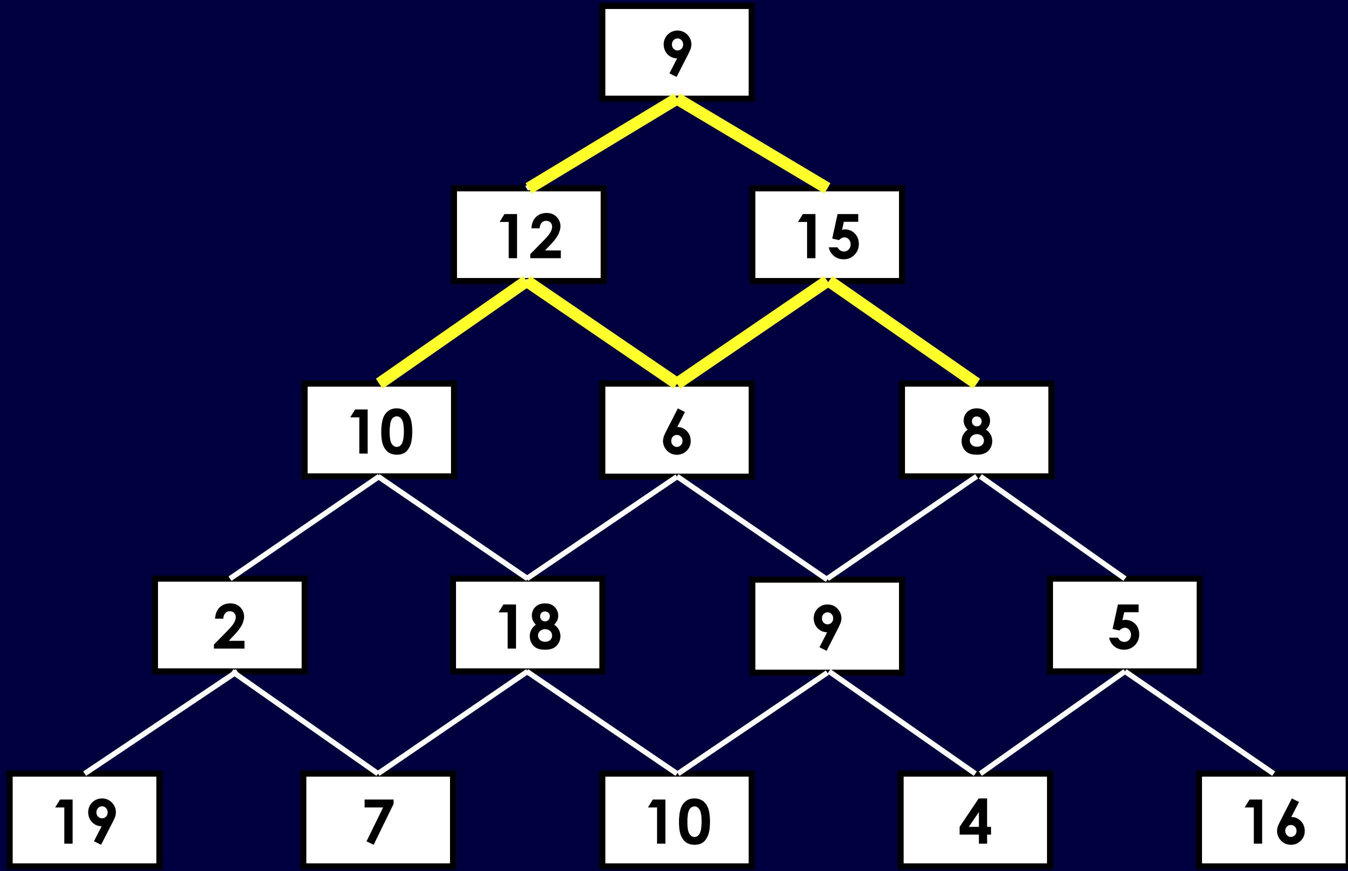
枚举法求解

- 思路2：枚举算法。

从顶层到第2层有2种选择

从顶层到第3层有2^2即四种可能路径

如果目标层数是较大的n层，将有2^{n-1} 条可能路径，需要列出的路径数太多



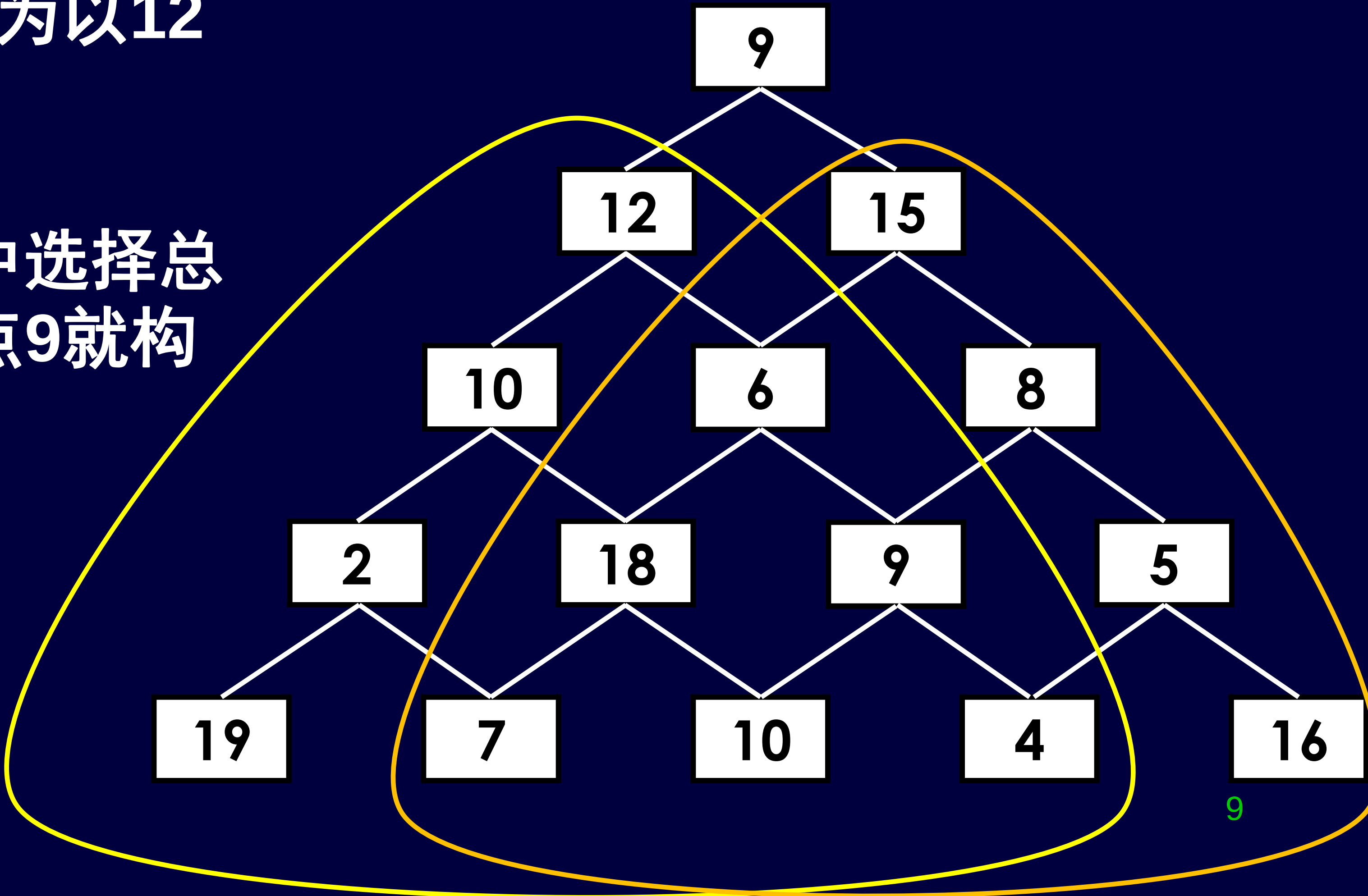
分治法求解

❖思路3：分治算法。

以9为顶的原问题可以分解为以12为顶和以15为顶的子问题。

求出子数塔的最优解，从中选择总和最大的最优解再加上节点9就构成了原问题的最优解。

分析数塔的结构，子问题的数量与枚举的数量级是相同的。



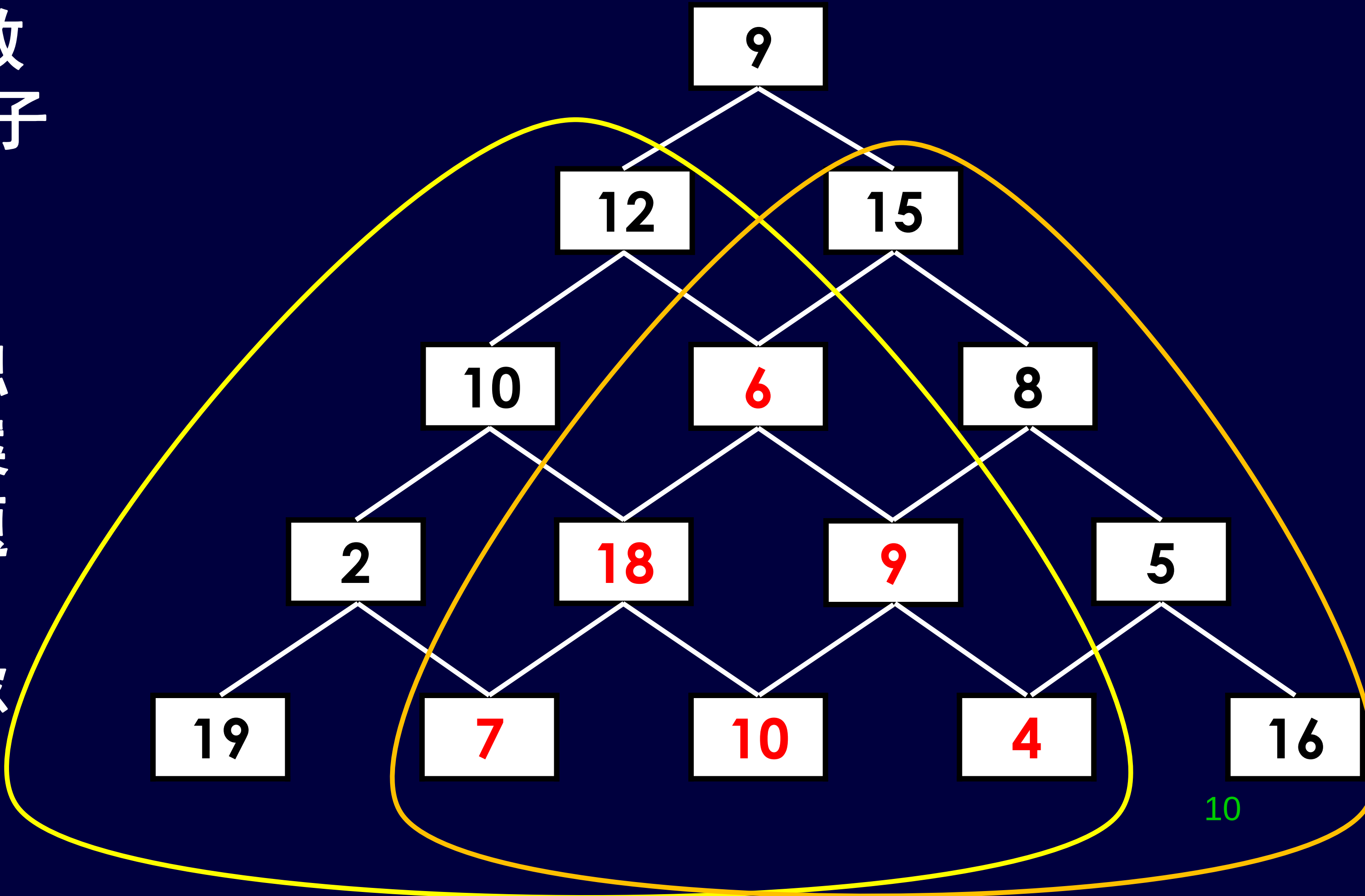
动态规划求解

❖思路4：动态规划。

以12为顶与以15为顶的子数塔共享了以6为顶的这个子子数塔。

由于划分出的子问题不是独立的，考虑将子问题的答案记录下来，保证每个子问题只被求解一次。

自下而上进行决策，先考虑第4层和第5层的9个数据。

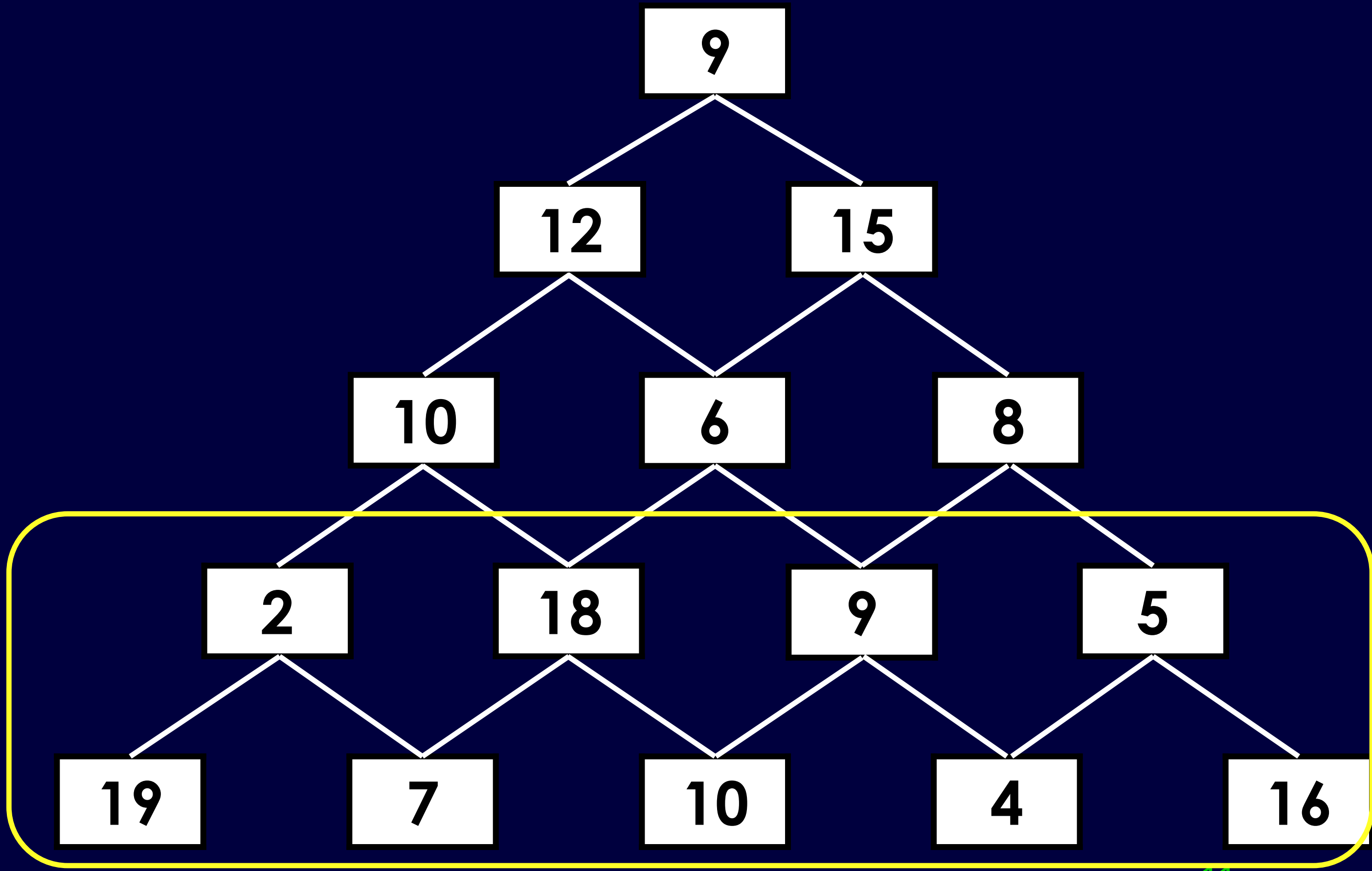


动态规划求解

❖思路4：动态规划。

以12为顶与以15为顶的子数塔共享了以6为顶的这个子子数塔。

由于划分出的子问题不是独立的，考虑将子问题的答案记录下来，保证每个子问题只被求解一次。
自下而上进行决策，先考虑第4层和第5层的9个数据。

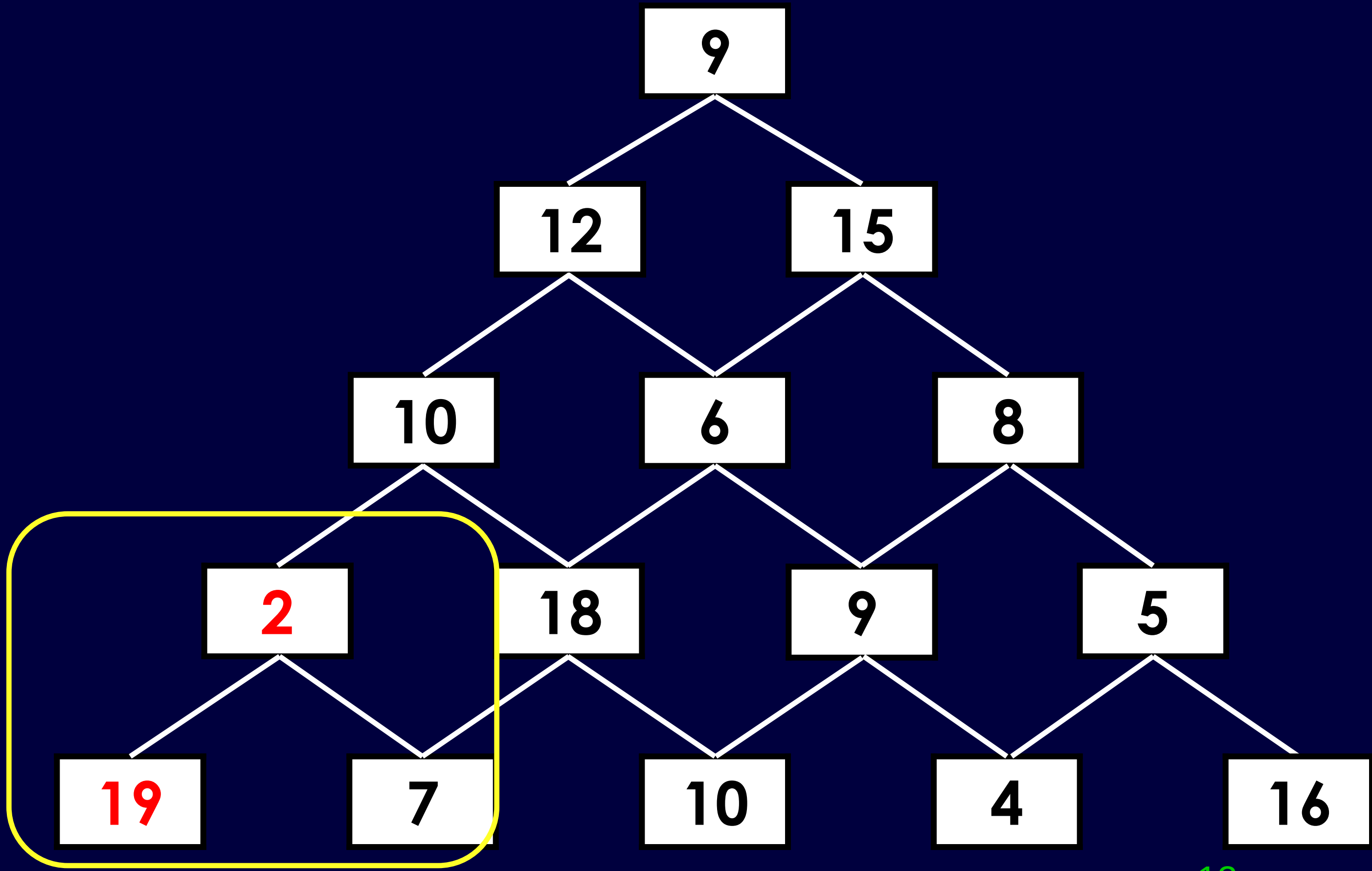


动态规划求解

根据第4层和第5层的9个数据，我们作如下判定：

① 如果最优的路径经过了第4层的2节点，那么它必然也经过第5层的19节点

理由： 第5层往下没有更低的层数，因此只需考虑当前的最大值，此时第5层仅有19和7两个节点可选，因此选择最大值19。



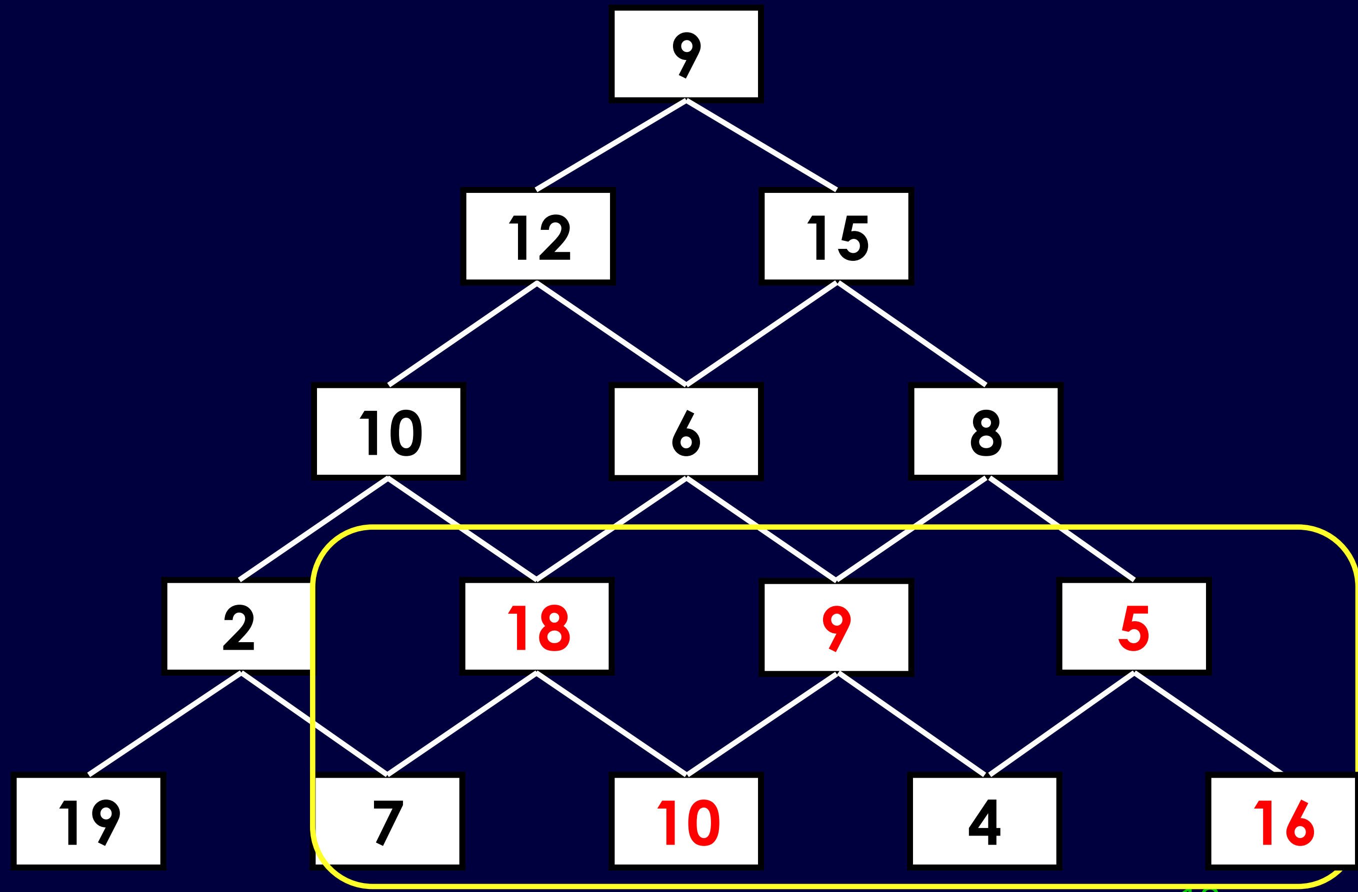
动态规划求解

同理可得：

② 经过了18节点的最优路径，必然也经过第5层的10节点，第四和第五层的值相加应该为18+10=28。

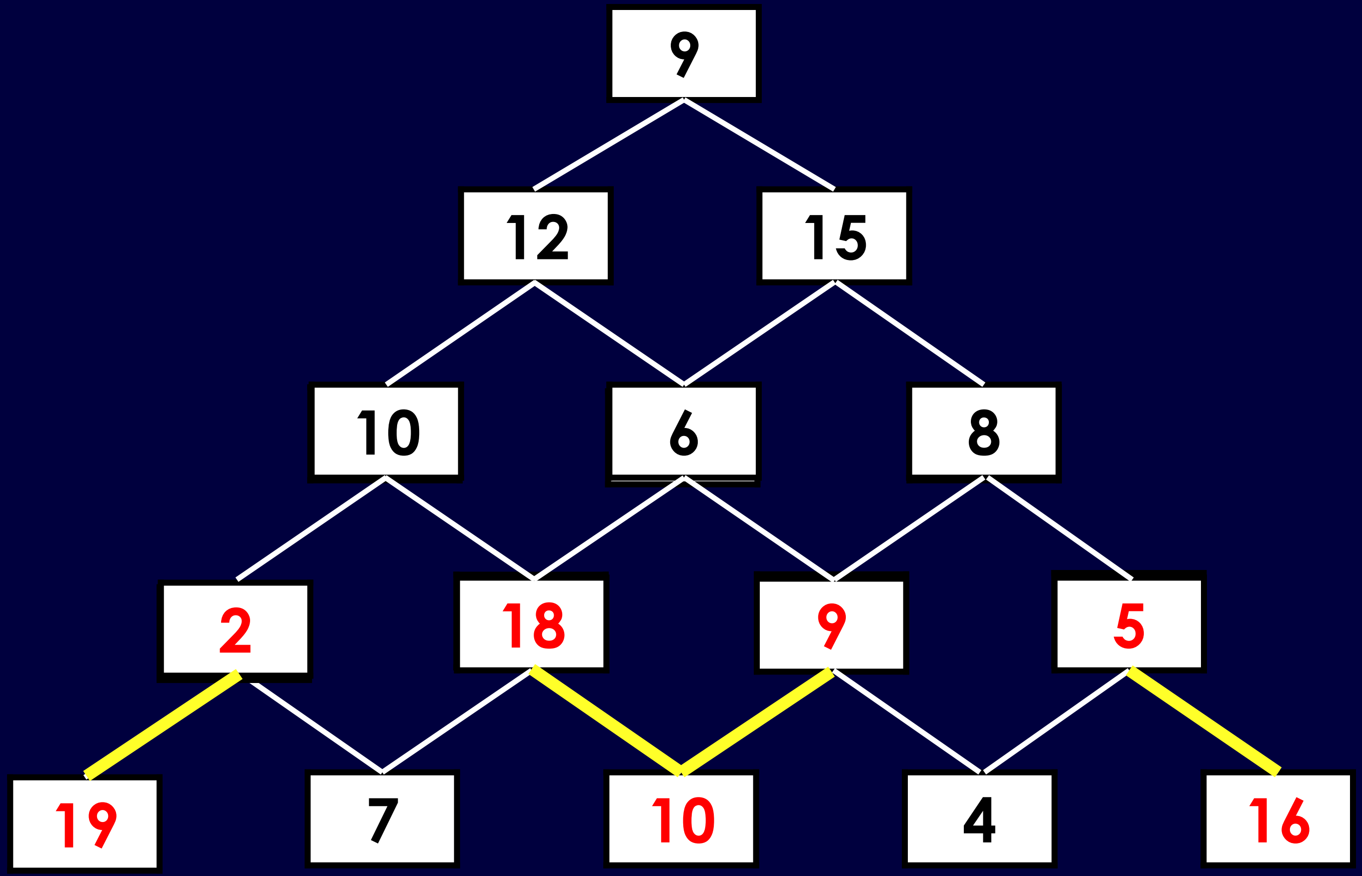
③ 9节点对应10节点，这两层的值相加为19。

④ 5节点对应16节点，这两层的值相加为21。



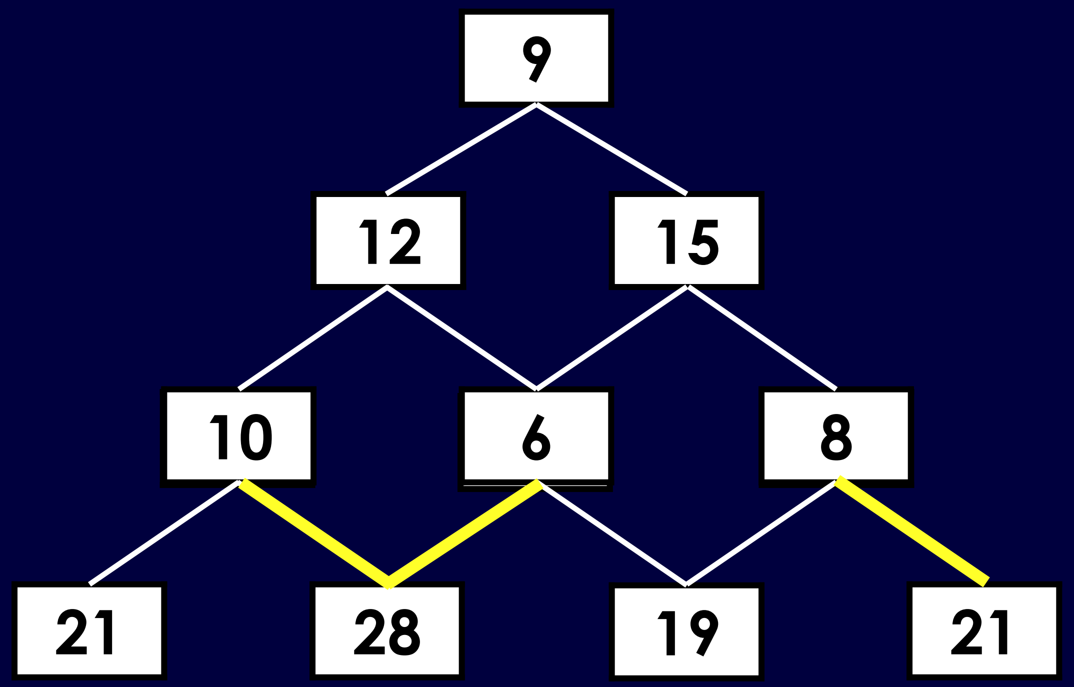
动态规划求解

❖**降阶：**此时，如果我们用这些值取代原第四层对应节点的值，并删掉第五层节点，就可以把一个五阶数塔转换成一个四阶数塔。



动态规划求解

❖继续降阶：利用同样的方法我们还可以将四阶数塔变成三阶。
我们进而一步步降阶，直到最后的1阶数塔，也就是这个问题的最优解。



实现的数据结构

*data*数组

9				
12	15			
10	6	8		
2	18	9	5	
19	7	10	4	16

d 数组

d[1,1]				
d[2,1]	d[2,2]			
d[3,1]	d[3,2]	d[3,3]		
d[4,1]	d[4,2]	d[4,3]	d[4,4]	
d[5,1]	d[5,2]	d[5,3]	d[5,4]	d[5,5]

❖**具体代码实现：**我们将使用一个二维数组$data$将原数塔存储为如左上图的三角阵形式，用二维数组d的每个节点$d[i,j]$存储第i行j列节点为顶的子数塔的最优解的值。则原问题的最优解为$d[1,1]$。

递归求解

*data*数组

9				
12	15			
10	6	8		
2	18	9	5	
19	7	10	4	16

d 数组

d[1,1]				
d[2,1]	d[2,2]			
d[3,1]	d[3,2]	d[3,3]		
d[4,1]	d[4,2]	d[4,3]	d[4,4]	
19	7	10	4	16

❖递归定义最优解的值，也就是要写出$d[i,j]$的递推公式。

从上面的分析不难发现：

$$d[i,j] = \max(d[i+1,j], d[i+1,j+1]) + data[i,j]$$

我们的动态规划算法需要从值已知的第n层开始，自底向上地计算数组d中各行的值。

总结

*data*数组

9				
12	15			
10	6	8		
2	18	9	5	
19	7	10	4	16

d 数组

59				
50	49			
38	34	29		
21	28	19	21	
19	7	10	4	16

最后，$d[1,1]$节点值就是我们所求的最优解，结束计算。

与贪心算法相比，动态规划算法在贪心的基础上增加了遍历的操作，保证所得到的的解释最优的。

动态规划算法每个阶段的决策都使得问题规模变小，加上对中间的结果的记录，使得该算法消除了枚举与分治对部分子问题的重复求解，优化了自身的效率。